- Function Complexity:

  1) code complexity:

     A) find the complexity of <u>return value</u>

```
5   int func(int n){
6       int temp = 0;
7       for(int i=n/2;i<=n;i++){
8           for(int j=2;j<=n;j*=2){
9               temp=temp+n/2;
10          }
11      }
12      return temp;
13  }
```

     B) find the number of repetitions of the main instruction in the following function

```
16  int func2(int n){
17      int i=n;
18      while (i>1){
19          int j=1;
20          while(j<n){
21              j=j*3;
22              i=i/2;
23              main_instruction
24          }
25      }
26  }
```

     C) find complexity of functions below

C-1)
```
for (int i = 0; i < n + 100; ++i) {
        for (int j = 0; j < i * n ; ++j){
            sum = sum + j;
        }
        for (int k = 0; k < n + n + n; ++k){
            c[k] = c[k] + sum;
        }
}
```

C-2)
```
for(i = n; i > 1; i = log(i)) {
        k = 0;
        for (j = 1; k < n; j++)
            k = k + j;
}
```

C-3)
```
for(i = n; i > 2; i = i^(1/5))
        for (int j = 1; j < n; j++)
            for (int k = 1; k < n; k++)
                j*=2;
```

2) logic of complexity

A) Provide a proof or violation of the following relations

$$f(n) \in O(s(n)),\ g(n) \in O(r(n)) \rightarrow \frac{f(n)}{g(n)} \in O(\frac{s(n)}{r(n)})$$

$$f(n) \in O(g(n)) \rightarrow g(n) \in \Omega(f(n))$$

$$log(f(n)) \in \Theta(log(g(n))) \rightarrow f(n) \in \Theta(g(n))$$

B) assume $h(n) = \lg^2 n,\ g(n) = \lg^{\lg n} n,\ f(n) = 4^{\lg n}$. then identify the correct and incorrect items

١. $f(n) \in \mathrm{O}(g(n)),\ f(n) \in \Omega(h(n))$

٢. $g(n) \in \Omega(h(n)),\ h(n) \in \Omega(f(n))$

٣. $f(n) \in \Theta(h(n)),\ g(n) \in \Omega(f(n))$

٤. $h(n) \in \mathrm{O}(g(n)),\ f(n) \in \Theta(g(n))$

٥. $g(n) \in \Omega\big(h(n)\big)\quad f(n) \in O(g(n))$

3) Sort functions

A) Sort the following functions according to their time complexity

A-1) $n2^n, \sum_{i=0}^{n} \frac{n^i}{i!}, n^{loglogn}, logn!, n^2, 2^n$

A-2) $n!, loglogn, loglog * n, log * logn, n^n, \sum_{i=0}^{n} \frac{n^i}{i!}$

A-3) $nloglogn,\ n^{2^n}, \sum_{i=0} i^3, n^n, logn^{logn},\ log * n$

B) Sort the following functions according to their time complexity

$$logn,\ n^n,\ \sum_{i=0}^{\frac{n}{2}} n - 2i,\ log(n!),\ 2^n,\ \sqrt[10]{n}$$

4)recursive complexity

A) find complexity of these recursive functions

A-1) $T(n) = 2T(n-1) + n$

A-2) $T(n) = T(\sqrt{n}) + O(\log(\log n))$

A-3) $T(n.k) = T\left(\frac{n}{2}.k\right) + T\left(n.\frac{k}{4}\right) + kn$

B) find complexity of these recursive functions

B-1) $T(n) = 7\,T\left(\frac{n}{7}\right) + \frac{n}{\log(n)}$

B-2) $T(n) = \frac{1}{5}T\left(\frac{n}{5}\right) + \frac{3}{5}T\left(\frac{3n}{5}\right) + n$

B-3) $T(n) = \sqrt{n}T(\sqrt{n}) + nlog(\log(n))$

C) find complexity of function

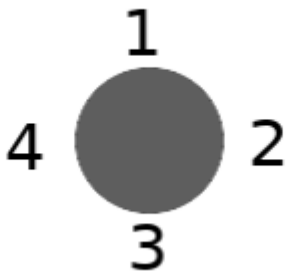
```
func2(n) {
     if n <= 1
          return;
     for (int i = 0; i < n*n; i++)
          print(i); // O(1)
     func2(n-2);
}
```

5)recursive algorithm

دور یک میز گرد n نفر نشسته‌اند. (n عددی زوج است) افراد دور میز می‌خواهند به هم دست بدهند ولی دست آن‌ها نباید از روی دست افراد دیگر رد شوند. (برای درک بهتر به توضیح تصویر توجه کنید)

الگوریتمی بازگشتی ارائه دهید که تعداد حالات مختلف دست دادن افراد دور میز را به شکلی که دست‌ها از روی هم عبور نکنند و هر نفر با یک نفر دیگر دست داده باشد را حساب کند. پیچیدگی زمانی این الگوریتم را حساب کنید.

توجه کنید که **نباید** از الگوریتم بازگشتی حافظه‌دار (Memoization) استفاده کنید. نیازی نیست که الگوریتم شما بهینه باشد و صرفا تمام حالات بررسی شوند.



برای مثال در تصویر بالا اگر فرد ۲ با ۴ و فرد ۱ با ۳ دست دهند، دست‌های آن‌ها از روی هم عبور می‌کنند و قابل قبول نیست. ولی اگر فرد ۱ با ۲ و فرد ۳ با ۴ دست دهند مشکلی ایجاد نمی‌شود. حالت ممکن دیگر دست دادن فرد ۱ با ۴ و فرد ۲ با ۳ است. در نتیجه برای این مثال ۲ حالت ممکن وجود دارد. برای ۶ نفر، ۵ حالت وجود دارد.